ReasonCache: Accelerating Large Reasoning Model Serving through KV Cache Sharing

Kaiwen Chen[1], Xin Tan[1], Minchen Yu[2], Jingzong Li[3], Hong Xu[1]
[1]The Chinese University of Hong Kong
[2]The Chinese University of Hong Kong, Shenzhen
[3]The Hang Seng University of Hong Kong

***Abstract*—Large Reasoning Models (LRMs) are integral to modern AI inference systems, but their long, auto-regressive reasoning processes incur substantial KV cache memory overhead that severely limits throughput and latency, degrading QoS for concurrent users. We observe that LRMs frequently generate highly similar intermediate reasoning steps corresponding to similar KV cache states across layers. Building on this insight, we propose ReasonCache, a novel KV cache management approach that uses a Collaborative Filtering Algorithm to efficiently identify reusable KV cache blocks and enables zero-copy cache reuse. Experiments demonstrate that ReasonCache achieves a peak throughput improvement of 89.2% and 40–60% average gains while maintaining higher accuracy than existing KV cache management techniques, leading to more responsive and cost-effective AI inference services.**

***Index Terms*—Large language models, Inference algorithms, Quality of service**

I. Introduction

Large Reasoning Models (LRMs), such as OpenAI's o1 [1], QwQ-32B [2] and DeepSeek-R1 [3], are increasingly powering AI inference systems with sophisticated reasoning capabilities [4]–[6]. To achieve this, LRMs perform extensive reasoning including long chain-of-thought (CoT) sequences, multi-strategy exploration, fine-grained problem decomposition, and self-verification [7], [8].

Meanwhile, Key-value (KV) caching stores attention states of preceding tokens to accelerate autoregressive generation, but incurs significant memory overhead that is particularly severe in LRMs due to their extensive reasoning. For instance, QwQ-32B can generate up to 13 redundant solutions for a trivial query such as "What is 2 + 3?", yielding a 1,953% token increase [8]. On AIME 24 [9], it produces 12,406 tokens per problem on average, requiring roughly 15 GB of GPU memory per request for the KV cache alone, severely limiting maximum concurrency.

Modern LLM inference consists of prefill and decoding stages [10], [11]. The decoding stage is memory-bound due to KV cache storage, and PagedAttention [12], [13] has become the dominant approach by partitioning KV cache into fixed-size blocks, enabling efficient memory management.

Recent advances in KV cache management exploit attention sparsity [14] to reduce memory overhead, categorized into eviction [15]–[20], selective loading [21], and merging [22], [23]. Their core idea is to estimate token importance via attention sparsity and selectively discard, load, or merge tokens. However, eviction is irreversible, risking permanent loss of tokens that may become important later; selective loading fails to reduce memory footprint; and merging can cause hallucinations due to inaccurate importance prediction. Critically, all these sparsity-based methods are fundamentally incompatible with PagedAttention [12], [13]: exploiting attention sparsity requires dynamic, fine-grained token- or layer-level operations, which disrupt contiguous block-based memory management [12], [13] and cause memory fragmentation.

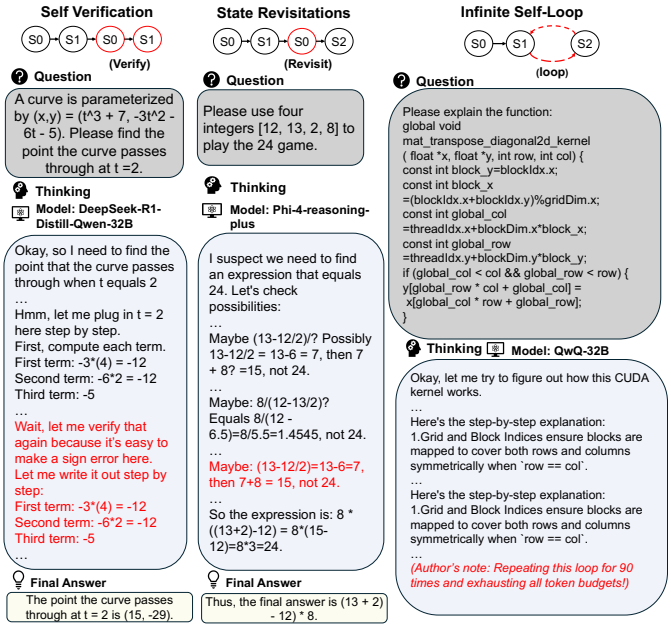


Fig. 1: Examples of *redundant thinking* patterns in Large Reasoning Models (LRMs).

Instead of attention sparsity, we identify a novel opportunity: LRMs frequently engage in *redundant thinking*—generating repetitive reasoning steps with highly similar content (self-verification, state revisitation, infinite self-loops, see Figure 1). These similar steps exhibit highly similar KV cache states across transformer layers, enabling memory reduction through KV block sharing without accuracy loss.

Two challenges must be overcome. First, reusable blocks must be identified accurately and efficiently—naive pairwise comparison is infeasible ($O(n^2)$). Second, reuse must integrate seamlessly into the decoding process without additional GPU

979-8-3195-1703-6/26/$31.00 ©2026 IEEE

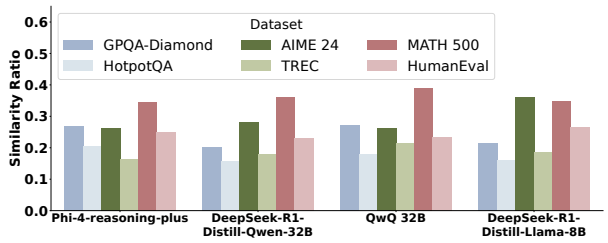


Fig. 2: Redundant thinking across different reasoning models and datasets.

memory bandwidth overhead.

To address these challenges, we introduce ReasonCache, a system that efficiently identifies reusable KV cache blocks using a two-stage, coarse-to-fine filtering approach. To identify similar blocks, ReasonCache first performs a lightweight, step-level comparison to prune the search space. It then employs a fine-grained, block-level similarity check, measuring the Euclidean distance between key and value vectors to accurately confirm reusability. For seamless integration, ReasonCache is designed at the block level, aligning with the memory management schemes of modern serving frameworks like vLLM [12] and SGLang [13].

To validate ReasonCache, we conduct comprehensive evaluations. Results show that ReasonCache achieves an average throughput improvement of 40–60% over dense attention while retaining over 98% accuracy, consistently outperforming existing strategies like StreamingLLM [16], SnapKV [17], and Quest [21] under comparable KV cache ratios.

We summarize our key contributions as follows: (1) We show that LRMs frequently generate highly similar intermediate reasoning steps (*redundant thinking*), indicating much KV cache content is similar. (2) We introduce ReasonCache, a novel KV cache management method that efficiently identifies and reuses similar KV cache blocks. (3) Comprehensive evaluations demonstrate up to 89.2% throughput gains while maintaining model accuracy.

II. Motivation

In this section, we first show that redundant thinking is common in LRMs, then examine its impact on KV cache patterns. Our two key observations are as follows:

Observation 1: *Redundant thinking in LRMs.* LRMs frequently generate repetitive reasoning steps, a phenomenon we term *redundant thinking*. Figure 1 illustrates three patterns: self-verification, state revisitation, and infinite self-loops. To quantify this, we compute cosine similarity between reasoning steps using bag-of-words vectors, defining the **similarity ratio** as the proportion of steps exceeding 0.8 similarity. Figure 2 shows 15–40% of content is redundant across models and benchmarks.

Observation 2: *Similar KV cache patterns from redundant thinking.* We hypothesize that redundant thinking manifests as similar KV cache states [23], [24]. Analyzing block-wise Euclidean distances, Figure 3 shows lexically similar tokens (red stars) have smaller distances (blue regions). Approximately

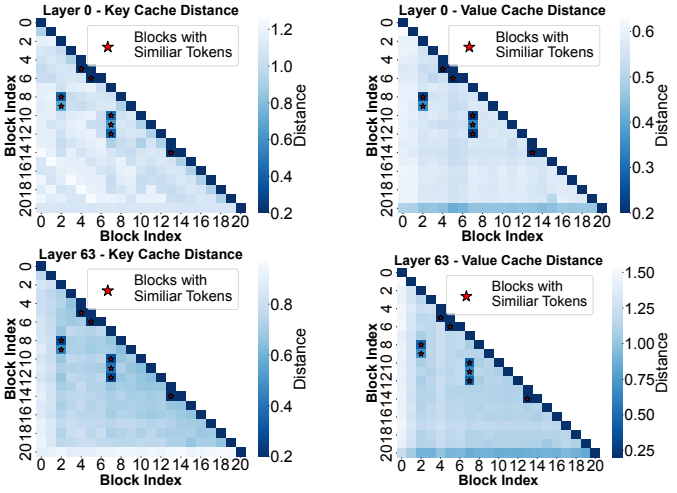


Fig. 3: Block-wise Euclidean distance heatmaps for KV cache (QwQ-32B on MATH 500).

38% of blocks have a lexically similar counterpart with low Euclidean distance, indicating reuse potential.

These observations motivate reusing similar KV cache blocks to reduce memory overhead.

III. ReasonCache

In this section, we introduce ReasonCache, a simple yet effective method designed to alleviate memory-bound constraints of LRM.

As illustrated in Figure 4, ReasonCache integrates two core components: Collaborative Filtering Algorithm for efficiently and accurately identifying reusable KV cache blocks and Zero-Copy KV Sharing mechanism for seamless reuse. The Collaborative Filtering Algorithm employs a two-stage process. It begins with a fast lexical and structural comparison of reasoning steps to identify potential candidates, followed by a precise Euclidean distance calculation on their corresponding KV cache blocks to confirm reusability (Section III-A). To implement the reuse, the KV sharing mechanism enables zero-copy sharing by integrating with vLLM's memory manager. It uses reference counting to map multiple logical blocks to a single physical copy, thereby avoiding costly data transfers (Section III-B). Together, these components enable memory-efficient decoding without compromising accuracy. We also provide theoretical justification for our design (Section III-C).

A. Collaborative Filtering Algorithm

The Collaborative Filtering Algorithm asynchronously runs on the CPU and identifies reusable blocks through a two-stage process.

*1) **Stage 1: Step-Level Similarity Filtering**:* The first stage performs coarse-grained filtering via lexical and structural analysis.

Lexical Similarity Pre-filtering: Each reasoning step is converted to a bag-of-words vector from the model's tokenizer, and cosine similarity with a length penalty is computed:

$$\text{sim}(S_c, S_k) = \frac{\mathbf{v}_c \cdot \mathbf{v}_k}{\|\mathbf{v}_c\|\|\mathbf{v}_k\|} \times \frac{\min(|S_c|, |S_k|)}{\max(|S_c|, |S_k|)}$$

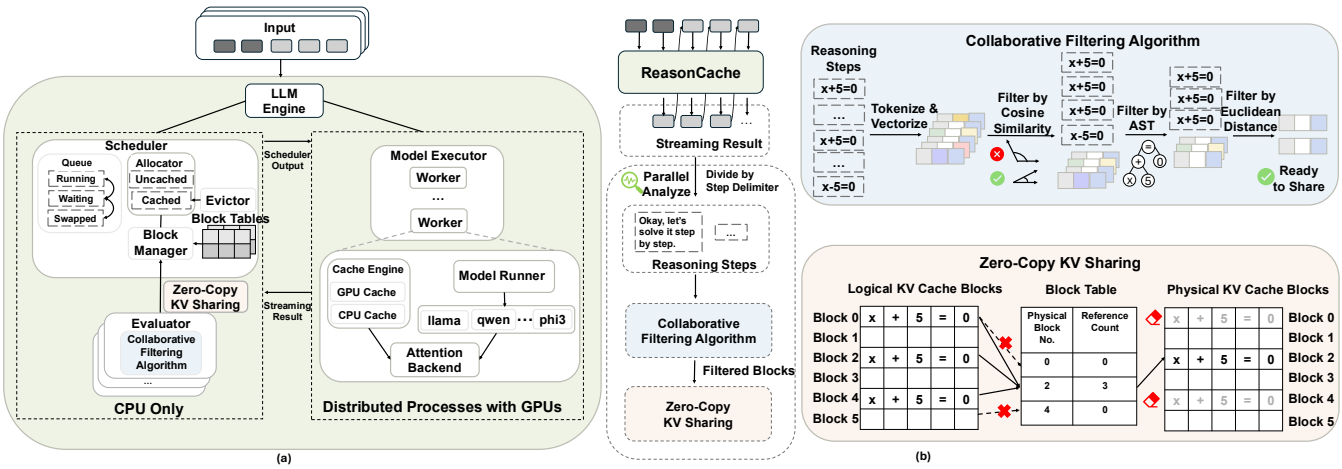


Fig. 4: System architecture (a) and workflow (b) of ReasonCache. While the LRM performs streaming token generation, ReasonCache executes Collaborative Filtering and Zero-Copy KV Sharing in parallel, reducing memory consumption and improving efficiency.

where $|S|$ is the character length of a step. A candidate is retained if $\text{sim}(S_c, S_k) > \tau$, where $\tau = \tau_{\text{strict}} - (\tau_{\text{strict}} - \tau_{\text{soft}}) \times \mu_{\mathcal{H}c}$ adaptively adjusts based on historical similarity scores $\mu_{\mathcal{H}c}$: when the model is in a repetitive state, τ lowers to promote more reuse; when exploring novel paths, τ rises for stricter filtering.

Structural Equivalence Verification: For formal languages (e.g., code, math), steps that pass the lexical filter are parsed into Abstract Syntax Trees (ASTs) [25] and compared via Tree Edit Distance (TED) [26]. A candidate is discarded if TED > 0, as this indicates structural mismatch. For unstructured text, this step is bypassed.

*2) **Stage 2: Block-Level Distance Filtering**:* Candidates passing Stage 1 undergo fine-grained Euclidean distance computation on their KV cache blocks, combined with Percentile-based Adaptive Thresholding (PAT).

Distance Computation: For blocks $\mathcal{B}_c$ and $\mathcal{B}_k$, the normalized cross-layer distance is:

$$\bar{d}(\mathcal{B}_c, \mathcal{B}_k) = \frac{1}{N} \sum_{i=1}^{N} \frac{\|\mathbf{K}_i[\mathcal{B}_c] - \mathbf{K}_i[\mathcal{B}_k]\|_2 + \|\mathbf{V}_i[\mathcal{B}_c] - \mathbf{V}_i[\mathcal{B}_k]\|_2}{2dh}$$

where N is the number of layers, d is block size, and h is the number of KV heads. To amortize cost, we expand $\|\mathbf{x}_c - \mathbf{x}_k\|^2 = \|\mathbf{x}_c\|^2 + \|\mathbf{x}_k\|^2 - 2\mathbf{x}_c^\top \mathbf{x}_k$, caching squared norms for reuse so that only dot products need recomputation.

Percentile-based Adaptive Thresholding (PAT): PAT sets a dynamic threshold based on the p-th percentile of the observed distance distribution, making it robust across models without fixed values. During warmup, distances are collected to establish a baseline; afterwards the p-th percentile serves as the reuse threshold.

Complexity: The effective time complexity is $O(n \log n)$, where n is sequence length. The two-stage filtering ensures the candidate set k grows logarithmically with n, and the asynchronous CPU offloading hides overhead from GPU inference.

B. Zero-Copy KV Sharing

As shown in Figure 4, sharing operates at the block level using PagedAttention's reference counting: when the evaluator identifies a reusable block, its reference count is incremented and the block table is updated, avoiding data copies. A sealed block is freed only when its reference count reaches zero. This leverages vLLM's decoupled CPU-GPU architecture, where CPU-managed block table updates enable instant reuse without GPU memory bandwidth overhead.

C. Theoretical Foundation

We provide a bounded analysis showing that KV cache sharing does not adversely affect accuracy. Let q_t be the query vector at step t, and let k_j, v_j denote the original key and value vectors. If k_j is replaced by k'_j with $\|k_j - k'_j\|_2 < \varepsilon$, then by Cauchy-Schwarz the attention score change satisfies $|q_t k_j^\top - q_t {k'_j}^\top| \leq \|q_t\|_2 \varepsilon$. Since softmax is Lipschitz continuous [27], [28], the attention weights are minimally perturbed. Similarly, replacing v_j with v'_j where $\|v_j - v'_j\|_2 < \delta$ bounds the output perturbation by δ. Thus, sufficiently close KV cache blocks induce negligible changes in model output, justifying the sharing approach.

IV. Evaluation

A. Settings

Benchmarks: We evaluate on HumanEval [29] (code), HotpotQA [30] and TREC [31] from LongBench [32] (long-context), AIME [9] and MATH-500 [33] (math), and GPQA Diamond [34] (science).

Models: We employ DeepSeek-R1-Distill-Qwen-32B, QwQ-32B, and Phi-4-reasoning-plus.

Baselines: We compare against Dense, StreamingLLM [16], Quest [21], and SnapKV [17].

Parameters: We use $\tau_{\text{strict}} = 0.9$, $\tau_{\text{soft}} = 0.7$, and PAT with $p = 80$.

Metrics: We report Accuracy, Throughput (tokens/s), and Average Memory Saving Ratio (AMSR). For sharing/eviction/merging methods, AMSR reflects physical KV

TABLE I: Accuracy and throughput (tokens/s) evaluation. Bold values represent the throughput improvement relative to Dense.

Model	Method	*Code*		*Multi-doc QA*		*Summarization*		*Math*						*Science*		Avg Acc	Avg throughput
		HumanEval		HotpotQA		TREC		AIME 2024		AIME 2025		MATH 500		GPQA Diamond			
		Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput		
DeepSeek R1 Distill Qwen 32B	ReasonCache	0.969	**(+62.36%)** 246.01	0.36	**(+42.18%)** 542.43	0.61	**(+58.87%)** 416.84	0.7	**(+72.38%)** 123.4	0.5	**(+43.1%)** 82.34	0.918	**(+89.2%)** 194.5	0.607	**(+56.6%)** 176.21	0.665	**(+56.33%)** 254.53
	Random	0.622	**(+63.09%)** 247.11	0.18	**(+40.98%)** 537.85	0.28	**(+54.89%)** 406.40	0.367	**(+68.73%)** 120.8	0.2	**(+42.2%)** 81.81	0.6	**(+85.9%)** 191.2	0.323	**(+53.8%)** 172.91	0.367	**(+54.25%)** 251.15
	Dense	0.982	151.52	0.375	381.51	0.62	262.38	0.733	71.59	0.533	57.55	0.927	102.82	0.631	112.40	0.686	162.82
QwQ 32B	ReasonCache	0.972	**(+72.31%)** 193.52	0.41	**(+42.34%)** 423.23	0.652	**(+61.23%)** 287.59	0.733	**(+78.27%)** 85.3	0.633	**(+68.9%)** 82.3	0.872	**(+81.6%)** 152.31	0.581	**(+59.0%)** 125.67	0.693	**(+59.29%)** 192.85
	Random	0.598	**(+70.67%)** 191.68	0.22	**(+41.89%)** 421.89	0.333	**(+60.89%)** 286.98	0.3	**(+79.1%)** 85.7	0.233	**(+72.6%)** 84.1	0.588	**(+82.7%)** 153.23	0.278	**(+55.9%)** 123.24	0.364	**(+58.92%)** 192.40
	Dense	0.968	112.31	0.45	297.34	0.648	178.37	0.766	47.85	0.633	48.72	0.894	83.85	0.611	79.06	0.71	121.07
Phi 4 reasoning plus	ReasonCache	0.945	**(+51.23%)** 338.33	0.4	**(+32.45%)** 695.88	0.61	**(+42.13%)** 556.84	0.8	**(+45.85%)** 245.21	0.633	**(+44.7%)** 243.21	0.903	**(+65.5%)** 384.21	0.508	**(+39.0%)** 243.22	0.686	**(+43.3%)** 386.7
	Random	0.482	**(+50.97%)** 337.75	0.21	**(+33.21%)** 699.87	0.315	**(+40.89%)** 552.08	0.333	**(+40.77%)** 243.2	0.133	**(+42.4%)** 239.41	0.492	**(+64.2%)** 381.23	0.203	**(+38.1%)** 241.81	0.324	**(+42.69%)** 385.05
	Dense	0.933	223.72	0.43	525.39	0.635	391.85	0.8	172.77	0.677	168.12	0.893	232.14	0.520	175.04	0.698	269.86

cache reduction; for selective loading methods like Quest [21], it measures the fraction of skipped KV blocks during attention. **Environment:** Experiments run on NVIDIA A800 GPUs (80GB) using vLLM 0.8.2, with 5 independent runs averaged.

B. Main Results

As shown in Table I, ReasonCache delivers 40–60% average throughput gains across all models and benchmarks with minimal accuracy loss. For instance, Phi-4-reasoning-plus achieves a 43.3% throughput boost while retaining 98.3% of Dense Attention accuracy.

Comparison with Other KV Cache Management Methods: Under comparable AMSR, ReasonCache consistently outperforms all baselines in accuracy (Table II). SnapKV shows lower accuracy on math/science tasks as it is optimized for long prompts rather than long decoding. StreamingLLM achieves moderate accuracy but loses information outside its sliding window.

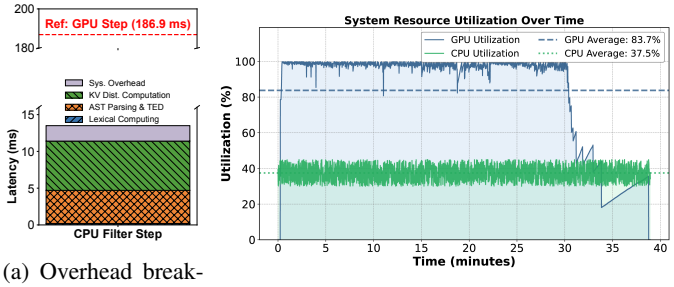
C. Overhead Analysis

We quantify ReasonCache's system overhead, focusing on CPU-intensive filtering. Using DeepSeek-R1-Distill-Qwen-32B on HumanEval, Figure 5a reports per-stage latency. ReasonCache introduces a modest 13.5 ms per step versus 186.9 ms for token generation. Since ReasonCache operates asynchronously on CPU, this overhead is completely masked by GPU inference.

Figure 5b presents CPU/GPU utilization under the same workload. CPU utilization stays within 35–45%, far from saturation, while GPU utilization sustains over 95% during the initial 30-minute peak, confirming our asynchronous offloading effectively decouples CPU from GPU inference. The subsequent GPU utilization drop reflects gradual task completion.

D. Ablation Study

Algorithm Design: We ablate three configurations: (1) Ours w/ only Cos (cosine similarity only); (2) Ours w/ Cos+AST (adding structural verification); and (3) Ours (full pipeline with Euclidean distance). Table III shows our full algorithm achieves the highest accuracy, though with slightly lower throughput due to more conservative sharing.



(a) Overhead breakdown.

(b) CPU/GPU utilization.

Fig. 5: Overhead breakdown per step and system utilization analysis.

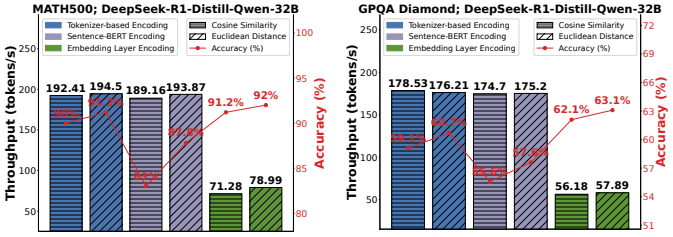


Fig. 6: Performance comparison of similarity measurement algorithms. Algorithms are categorized in step-level (distinguished by colors) and block-level (distinguished by line patterns).

Similarity Measurement Methods: Figure 6 compares step-level similarity algorithms: (1) Tokenizer-based Encoding (our default bag-of-words), (2) Sentence-BERT [35] (384-dim embeddings), and (3) native embedding layer. Tokenizer-based Encoding with Block-Level Euclidean Distance yields the highest accuracy because Euclidean distance on KV cache states [36] inherently captures the model's semantic structure, rendering external semantic models redundant and misaligned.

TABLE II: Evaluation of different KV cache management strategies. Average Memory Saving Ratio (AMSR) quantifies the memory efficiency of a strategy. For each benchmark, the highest and second-highest accuracies are highlighted in bold and underlined, respectively.

Model	Method	*Code*		*Multi-doc QA*		*Summarization*		*Math*						*Science*		Avg Acc	AMSR
		HumanEval		HotpotQA		TREC		AIME 2024		AIME 2025		MATH 500		GPQA Diamond			
		Acc	AMSR	Acc	AMSR	Acc	AMSR	Acc	AMSR	Acc	AMSR	Acc	AMSR	Acc	AMSR		
DeepSeek R1 Distill Qwen 32B	StreamingLLM	0.884	20.1%	0.235	15.8%	0.47	15.5%	0.567	22.34%	0.333	20.7%	0.764	38.2%	0.409	16.5%	0.523	21.31%
	SnapKV	0.872	9.6%	0.285	16.22%	0.51	16%	0.5	1.79%	0.2	1.8%	0.44	2.79%	0.293	8.2%	0.443	8.06%
	Quest	0.866	20.4%	0.24	16.02%	0.48	15.8%	0.6	20.98%	0.333	25%	0.5	39.9%	0.561	13.8%	0.511	21.7%
	ReasonCache	0.969	21.5%	0.36	16.51%	0.61	15.9%	0.7	21.18%	0.5	23.2%	0.918	39.1%	0.607	15.54%	0.665	21.85%
	Dense	**0.982**	0%	**0.375**	0%	**0.62**	0%	**0.733**	0%	**0.533**	0%	**0.927**	0%	**0.631**	0%	**0.686**	0%
QwQ 32B	StreamingLLM	0.799	21.7%	0.282	16.51%	0.49	17.3%	0.567	20.3%	0.367	20.6%	0.672	29.7%	0.530	17.5%	0.523	20.52%
	SnapKV	0.756	6.2%	0.32	15.7%	0.61	15.5%	0.1	1%	0.233	1.43%	0.176	1.55%	0.146	4.27%	0.334	6.52%
	Quest	0.915	20.8%	0.36	15.82%	0.56	17.8%	0.666	25.1%	0.533	23.5%	0.506	28%	0.535	18.1%	0.582	21.30%
	ReasonCache	**0.972**	22.1%	0.41	16.12%	**0.652**	18.1%	0.733	23.41%	**0.633**	23.1%	0.872	26.89%	0.581	18.12%	0.693	21.12%
	Dense	0.968	0%	**0.45**	0%	0.648	0%	**0.766**	0%	**0.633**	0%	**0.894**	0%	**0.611**	0%	**0.71**	0%
Phi 4 reasoning plus	StreamingLLM	0.812	14.8%	0.278	14.8%	0.45	13.4%	0.567	16.6%	0.367	17%	0.522	23%	0.333	18.9%	0.476	16.93%
	SnapKV	0.793	11.5%	0.292	15.5%	0.47	13.7%	0.2	2.56%	0.166	3%	0.146	4.76%	0.253	7.29%	0.331	8.33%
	Quest	0.854	15.3%	0.29	15.75%	0.47	13.6%	0.567	18.5%	0.433	17.7%	0.518	21%	0.312	17.8%	0.478	17.09%
	ReasonCache	**0.945**	16.8%	0.4	16.2%	0.61	14.2%	**0.8**	16.4%	0.633	17.2%	**0.903**	21.41%	0.508	17.6%	0.686	17.12%
	Dense	0.933	0%	**0.43**	0%	**0.635**	0%	**0.8**	0%	**0.677**	0%	0.893	0%	**0.520**	0%	**0.698**	0%

TABLE III: Ablation study on different configurations of ReasonCache, comparing their accuracy and throughput (tokens/s).

Model	Setting	*Code*		*Multi-doc QA*		*Summarization*		*Math*		*Science*		Avg Acc	Avg throughput
		HumanEval		HotpotQA		TREC		MATH 500		GPQA Diamond			
		Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput	Acc	throughput		
DeepSeek R1 Distill Qwen 32B	Ours w/ only Cos	0.902	277.99	0.31	572.21	0.535	436.21	0.78	223.68	0.505	214.21	0.606	344.86
	Ours w/ only Cos+AST	0.933	255.85	0.31	570.64	0.53	439.35	0.822	210.06	0.556	198.23	0.63	334.83
	Ours	**0.969**	246.01	**0.36**	542.43	**0.61**	416.84	**0.918**	194.5	**0.607**	176.21	**0.692**	315.198
QwQ 32B	Ours w/ only Cos	0.915	216.74	0.375	440.28	0.585	303.82	0.752	181.25	0.48	142.12	0.621	256.84
	Ours w/ only Cos+AST	0.939	201.26	0.375	444.39	0.59	301.39	0.8	167.54	0.532	133.46	0.647	249.61
	Ours	**0.972**	193.52	**0.41**	423.23	**0.652**	287.59	**0.872**	152.31	**0.581**	125.67	**0.697**	236.464
Phi 4 reasoning plus	Ours w/ only Cos	0.884	375.55	0.365	728.21	0.54	578.92	0.792	430.24	0.404	278.12	0.597	478.21
	Ours w/ only Cos+AST	0.939	358.63	0.37	730.67	0.54	580.78	0.85	412.32	0.449	252.52	0.63	466.98
	Ours	**0.945**	338.33	**0.4**	695.88	**0.61**	556.84	**0.903**	384.21	**0.508**	243.22	**0.673**	443.696

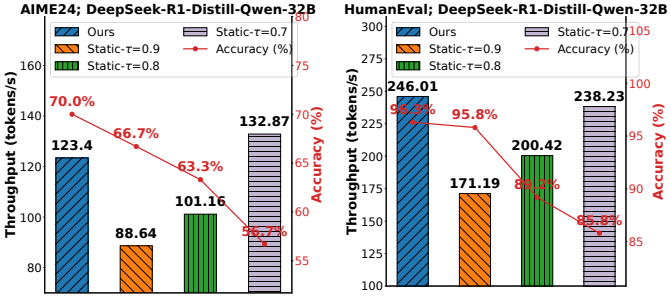


Fig. 7: Comparison of dynamic and fixed thresholding.

Impact of threshold: Figure 7 shows our dynamic threshold τ consistently outperforms fixed thresholds. Table IV further indicates PAT with $p = 80$ provides the optimal accuracy–throughput balance across all models and datasets.

TABLE IV: Performance comparison of different percentile values (p) for Percentile-based Adaptive Thresholding (PAT).

Model	Threshold	HumanEval		TREC		MATH 500	
		Acc	throughput	Acc	throughput	Acc	throughput
DeepSeek R1 Distill Qwen 32B	PAT-p=20	**0.97**	192.12	0.6	322.2	**0.924**	121.89
	PAT-p=50	0.957	212.34	0.605	352.13	0.922	149.21
	PAT-p=80	0.969	246.01	**0.61**	416.84	0.918	194.5
	PAT-p=100	0.933	255.85	0.53	439.35	0.822	210.06
QwQ-32B	PAT-p=20	0.97	132.89	0.645	215.27	**0.878**	101.23
	PAT-p=50	**0.976**	158.35	**0.655**	248.28	0.868	123.37
	PAT-p=80	0.972	193.52	0.652	287.59	0.872	152.31
	PAT-p=100	0.939	201.26	0.59	301.39	0.8	167.54
Phi 4 reasoning plus	PAT-p=20	**0.951**	261.23	**0.635**	412.3	0.893	310.2
	PAT-p=50	0.948	295.71	0.62	478.21	0.896	342.3
	PAT-p=80	0.945	338.33	0.61	556.84	**0.903**	384.21
	PAT-p=100	0.939	358.63	0.54	580.78	0.85	412.32

V. Related Work

Existing KV cache management falls into three categories: eviction (H2O [15], SnapKV [17], StreamingLLM [16], Lethe [18], RLKV [20]), selective loading (Quest [21]), and merging (MiniCache [22], D2O [23]). All rely on attention sparsity to estimate token importance. However, eviction is irreversible [37], selective loading retains memory footprint, and merging risks hallucinations from inaccurate importance prediction [38]. ReasonCache fundamentally differs by identifying reusable KV blocks through similarity rather than sparsity.

VI. Conclusion

We present ReasonCache, a novel KV cache management approach that efficiently identifies and reuses similar KV cache blocks. Comprehensive evaluations show that ReasonCache achieves a peak throughput improvement of 89.2% and an

average gain of 40-60%, while reducing KV cache memory usage by 17-21%, all without compromising accuracy.

ACKNOWLEDGMENTS

This work is partly supported by funding from CUHK (4937007, 4937008, 5501329, 5501517) and Huawei (7010881).

REFERENCES

[1] A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney *et al.*, "Openai o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[2] Qwen, "Qwq-32b: Embracing the power of reinforcement learning," 2025. [Online]. Available: https://qwen.ai/blog?id=qwq-32b

[3] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[4] W. Liu, X. Ma, W. Sun, Y. Zhu, Y. Li, D. Yin, and Z. Dou, "Reasonrank: Empowering passage ranking with strong reasoning ability," *arXiv preprint arXiv:2508.07050*, 2025.

[5] K. Zhao, F. Xu, and Y. Li, "Reason-to-recommend: Using interaction-of-thought reasoning to enhance llm recommendation," *arXiv preprint arXiv:2506.05069*, 2025.

[6] X. Zhou, G. Tie, G. Zhang, W. Wang, Z. Zuo, D. Wu, D. Chu, P. Zhou, L. Sun, and N. Zhenqiang Gong, "Large reasoning models in agent scenarios: Exploring the necessity of reasoning capabilities," *arXiv e-prints*, pp. arXiv–2503, 2025.

[7] F. Xu, Q. Hao, Z. Zong, J. Wang, Y. Zhang, J. Wang, X. Lan, J. Gong, T. Ouyang, F. Meng *et al.*, "Towards large reasoning models: A survey of reinforced reasoning with large language models," *arXiv preprint arXiv:2501.09686*, 2025.

[8] X. Chen, J. Xu, T. Liang, Z. He, J. Pang, D. Yu, L. Song, Q. Liu, M. Zhou, Z. Zhang *et al.*, "Do not think that much for 2+ 3=? on the overthinking of o1-like llms," *arXiv preprint arXiv:2412.21187*, 2024.

[9] MAA, "American invitational mathematics examination - aime," 2025. [Online]. Available: https://huggingface.co/datasets/di-zhang-fdu/AIME_1983_2024

[10] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[11] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 193–210.

[12] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[13] L. Zheng, L. Yin, Z. Xie, C. L. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Sglang: Efficient execution of structured language model programs," *Advances in Neural Information Processing Systems*, vol. 37, pp. 62 557–62 583, 2024.

[14] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[15] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.

[16] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[17] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 22 947–22 970, 2024.

[18] H. Zeng, D. Zhao, P. Yang, W. Hou, T. Zheng, H. Li, W. Ji, and J. Zhai, "Lethe: Layer-and time-adaptive kv cache pruning for reasoning-intensive llm serving," *arXiv preprint arXiv:2511.06029*, 2025.

[19] J. Hu, W. Huang, W. Wang, Z. Li, T. Hu, Z. Liu, X. Chen, T. Xie, and Y. Shan, "Raas: Reasoning-aware attention sparsity for efficient llm reasoning," *arXiv preprint arXiv:2502.11147*, 2025.

[20] W. Du, L. Jiang, K. Tao, X. Liu, and H. Wang, "Which heads matter for reasoning? rl-guided kv cache compression," *arXiv preprint arXiv:2510.08525*, 2025.

[21] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," in *Forty-first International Conference on Machine Learning*.

[22] A. Liu, J. Liu, Z. Pan, Y. He, R. Haffari, and B. Zhuang, "Minicache: Kv cache compression in depth dimension for large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 139 997–140 031, 2024.

[23] Z. Wan, X. Wu, Y. Zhang, Y. Xin, C. Tao, Z. Zhu, X. Wang, S. Luo, J. Xiong, L. Wang *et al.*, "D2o: Dynamic discriminative operations for efficient long-context inference of large language models," in *ICLR*, 2025.

[24] Z. Wang, B. Jin, Z. Yu, and M. Zhang, "Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks," *arXiv preprint arXiv:2407.08454*, 2024.

[25] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, *Compilers Principles, Techniques & Tools*. Pearson Education, 2007.

[26] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[27] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: Analysis and efficient estimation," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[28] L. Collins, A. Parulekar, A. Mokhtari, S. Sanghavi, and S. Shakkottai, "In-context learning with transformers: Softmax attention adapts to function lipschitzness," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 92 638–92 696.

[29] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[30] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 2369–2380.

[31] X. Li and D. Roth, "Learning question classifiers," in *The 19th International Conference on Computational Linguistics*, 2002.

[32] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024, pp. 3119–3137.

[33] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, vol. 1, 2021.

[34] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman, "Gpqa: A graduate level google-proof qa benchmark," 2023. [Online]. Available: https://arxiv.org/abs/2311.12022

[35] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Nov. 2019.

[36] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," *Advances in neural information processing systems*, vol. 35, pp. 17 359–17 372, 2022.

[37] Y. LI, H. Jiang, Q. Wu, X. Luo, S. Ahn, C. Zhang, A. H. Abdi, D. Li, J. Gao, Y. Yang *et al.*, "Scbench: A kv cache-centric analysis of long-context methods," in *The Thirteenth International Conference on Learning Representations*, 2025.

[38] J. Y. Yang, B. Kim, J. Bae, B. Kwon, G. Park, E. Yang, S. J. Kwon, and D. Lee, "No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization," *arXiv preprint arXiv:2402.18096*, 2024.